Welcome

Welcome to the workshop and to our online learning environment (Moodle). Please see the User Manual for details of the system.

This workbook will cover the following topics during the first week:

- Introducing the BlueJ IDE
- Intro to Object Oriented Programming (OOP)
  - what is an object?
  - constructors
  - data members and methods
  - interaction with an object
- Putting objects to work
  - simple assignment
  - methods
  - parameters
  - return values

By the end of the week you should be able to create a simple program.

Visual Aids

We have visual guides and jargon boxes to help you get to grips with the applications and many technical terms you must use.

Where you see this logo, we have Flash demos in the workshop course to help you visualize the workings of the examples.

The workshop also uses screencasts to walk you through some of the tasks.

Notice

This workbook is intended for use with the Java Programming Workshop run at http://ib-computing.net/.

Every effort has been made in the preparation of this workbook to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. The author/publisher will not be held liable for any damages caused or alleged to be caused directly or indirectly by this workbook.

This work is released under a creative commons license. There are legal restrictions on your use of the work. See the screenshot on the next page or go directly to: http://creativecommons.org/licenses/by-nc-sa/3.0/nz/

© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

creative commons

Attribution-Noncommercial-Share Alike 3.0 New Zealand

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial — You may not use this work for commercial purposes.

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

Waiver — Any of the above conditions can be **waived** if you get permission from the copyright holder.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or **fair use** rights;
- The author's **moral** rights;
- Rights other persons may have either in the work itself or in how the work is used, such as **publicity** or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

SOME RIGHTS RESERVED

© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

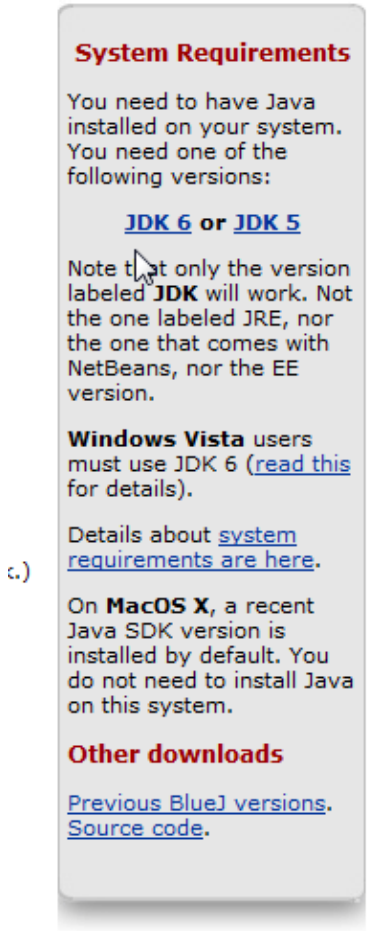
Introducing BlueJ

We will use the BlueJ Integrated Development Environment in this course. You do not absolutely have to use this, but things will be easier of you do. You will certainly need the JDK (Java Development Kit) as detailed below.

> ***Integrated Development Environment?***
> *Or IDE – this is a package of several tools to help programmers. It usually consists of an editor, some templates, a debugger and a help system.*
> *We will use many of these features in BlueJ.*

If you don't have the JDK

Or if you have an older version that does not work with the most recent BlueJ release


System Requirements
You need to have Java installed on your system. You need one of the following versions:
JDK 6 or JDK 5
Note that only the version labeled JDK will work. Not the one labeled JRE, nor the one that comes with NetBeans, nor the EE version.
Windows Vista users must use JDK 6 (read this for details).
Details about system requirements are here.
On MacOS X, a recent Java SDK version is installed by default. You do not need to install Java on this system.
Other downloads
Previous BlueJ versions.
Source code.

Notice that, according to the information on the BlueJ downloads page, MacOS X users will not need to install the JDK.

The JDK can be installed from the location:

http://java.sun.com/javase/downloads/index.jsp click the link near the top of the page:

Sun Java Solaris Communities My SDN Account Join SDN
Sun Developer Network (SDN)
APIs Downloads Products Support Training Participate
SDN Home > Java Technology > Java SE >
Java SE Downloads
It's time
Download the complete platform and runtime environment
» Get the JDK download
Overview Technologies Documentation Community Support Downloads
Latest Release | Next Release (Early Access) | Embedded Use | Real-Time | Previous Releases
Java SE Runtime Environment (JRE)
JRE 6 Update 13
This release includes the highly anticipated 64-bit Java Plug-In (for 64-bit
Download

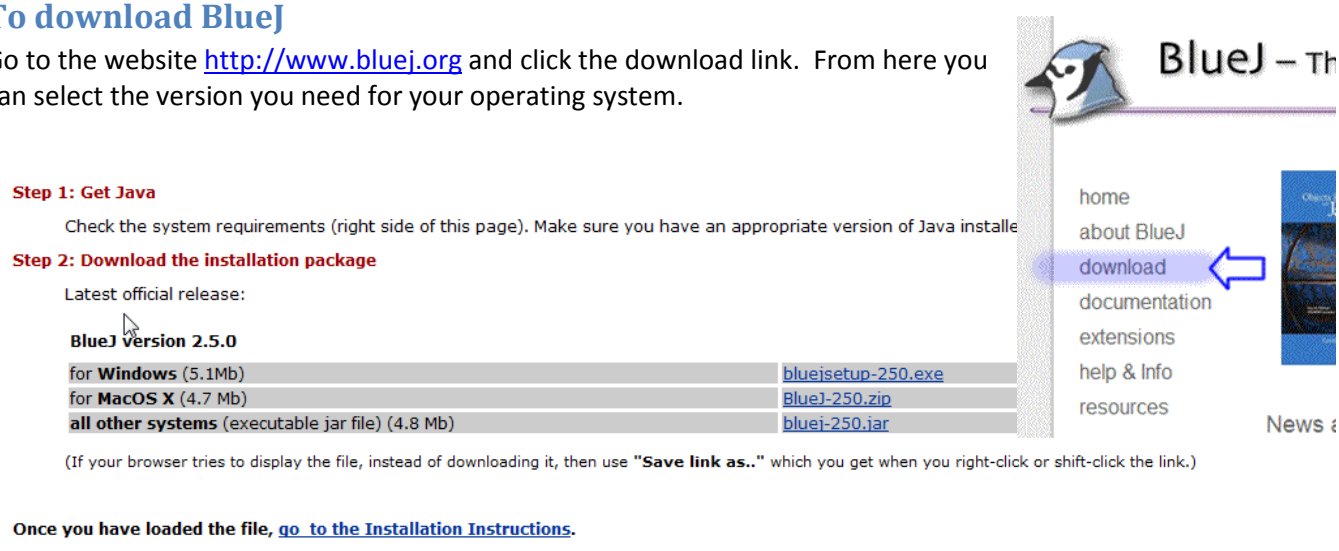
Select the Operating System you are using from the drop-down list, continue, agree and download the file.

Read the installation instructions for your operating system and install the software.

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

To download BlueJ

Go to the website http://www.bluej.org and click the download link. From here you can select the version you need for your operating system.


BlueJ – Th
home
about BlueJ
download
documentation
extensions
help & Info
resources
News
Step 1: Get Java
Check the system requirements (right side of this page). Make sure you have an appropriate version of Java installe
Step 2: Download the installation package
Latest official release:
BlueJ version 2.5.0
for Windows (5.1Mb) bluejsetup-250.exe
for MacOS X (4.7 Mb) BlueJ-250.zip
all other systems (executable jar file) (4.8 Mb) bluej-250.jar
(If your browser tries to display the file, instead of downloading it, then use "Save link as.." which you get when you right-click or shift-click the link.)
Once you have loaded the file, go to the Installation Instructions.

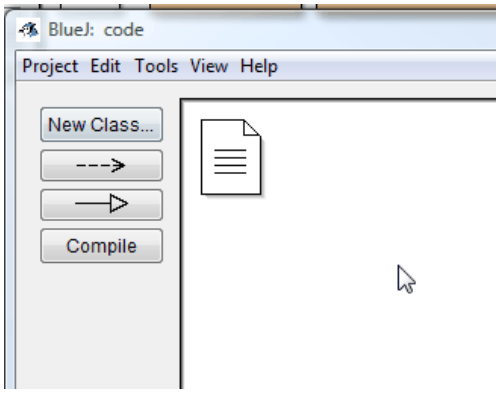
Double click (or drag to the applications folder) the downloaded file to install BlueJ (see the installation Instructions for further detailed information.

After it is installed you have an option to read the readme file and/or launch BlueJ itself.

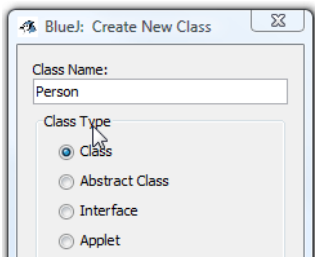
Starting with BlueJ

Normally, BlueJ will ask you to confirm which version of Java it is planning to use, you may want to browse for a newer version if you have just installed the latest JDK. Normally you can just accept the default.

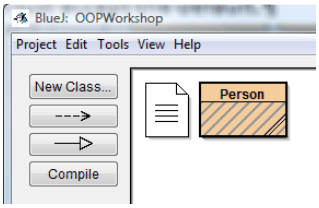
When you get to the main screen you will see this window:



You can click the New Class… button or right click (control click) in the window to get a popup menu of choices.



Select a new Class and enter the name Person, then click OK. You see a yellow box appear with the name Person in it.



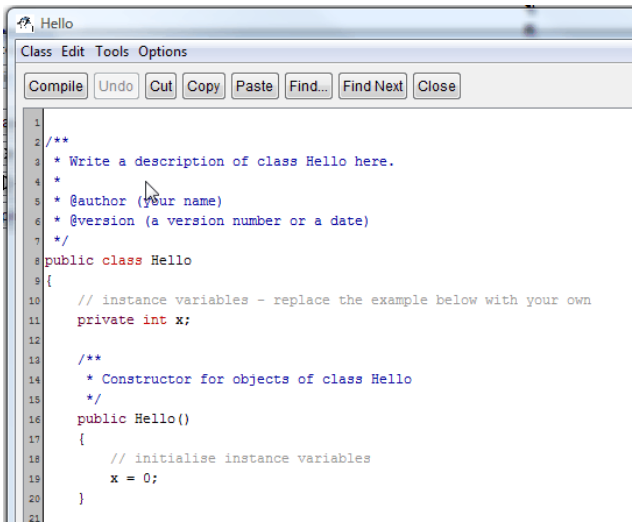
Double click the box labelled Person to see the code in the editor.

This is scary.

Don't panic.

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Introduction to Object Oriented Programming



The Code Editor

This editor is part of the IDE. A nice thing about it is the way it colour codes the different types of text for us. For example, the nice blues and greys are comments.

We will delete most of this code for now.

Machine code
Often thought of as 1's and 0's this is the language that computers actually run through their silicon chips.

Syntax error
A spelling mistake, like writing publik instead of public (or even Public instead of public).

BlueJ is useful for teaching as it does allow us to create our own templates similar to this one, but we won't do that just yet.

Let's first change the comments and then enter the Java code as follows:

```
public class Person
{
  String name = "Richard";
}
```

This is a class – the basis of pretty much everything in Java. It isn't very useful yet.

You can cut and paste the code from the workbook if you like, making sure to replace all of the existing code[1]. Or you can replace the bits that are different.

We will break everything down and define some jargon. Getting the jargon right is important as otherwise we can't communicate when we need help from an expert. Different experts have slightly different jargon, unfortunately, but we will try to be consistent.

Compilers and interpreters

These are computer programs that turn ordinary text into *machine code* (the stuff that computers actually run on the hardware).

The *compiler* in Java translates the source code (our Person file) into *bytecodes* - a compact form – which is, in turn, sent to an *interpreter* to be converted to machine code run on the computer.

If the compiler cannot turn the source code into bytecodes, an error is given (you will see a lot of these). This is known as a *syntax error*.

[1] Many word processors use "smart quotes" – if you cut and paste code from this document you may need to re-type these in BlueJ. Alternatively, you can get the source code from the code folder in Week 1 of the course.

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

To help you out a bit, we will put definitions of these terms in a word list every now and then. With that background, let's examine the code:

> ***class***
> *Java programs are made up of objects, the class defines the object.*
>
> ***variable***
> *A variable is an area of computer memory given a convenient name by the programmer.*

A class

```
public class Person
{
```

A class is a template (or blueprint) for object *instances*. Classes are generally public and, by convention, start with a capital letter (upper case), followed by small letters (lowercase). Public means that other classes can create instances from the template – you can actually do something with an instance.

The entire class code must appear between two braces, one opening and one closing:

```
{
}
```

It is conventional to line them up vertically (or sometimes horizontally). You will notice that after the opening brace, the source code is indented. This makes it more readable for humans.

Instance variable

```
private String name = "Richard";
```

An *instance variable* is a named location in computer memory which we can use to store different types of data. By convention, an instance variable always starts with a lowercase letter. In this case we are storing String data (a string of characters, alphabetic, numeric or other symbols).

The name of the instance variable is `name`. The `private` keyword means that it can only be accessed inside the Person class (this is good practice).

It is quite common to think of an instance variable as a box - with the difference that this type of box can only store one item of data:

Internal reference	Computer Memory	Named instance variable
01110101		
01110110	"Richard"	name
01110111		
01111000		
01111001	2176.50	balance
01111010		
01111011		

© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

As humans we find I much easier to keep track of things that have names rather than numbers. The computer tracks memory locations (and everything else) using binary machine code (we think of it as 1s and 0s). Thus the ability to name memory locations is very useful:

Data Types

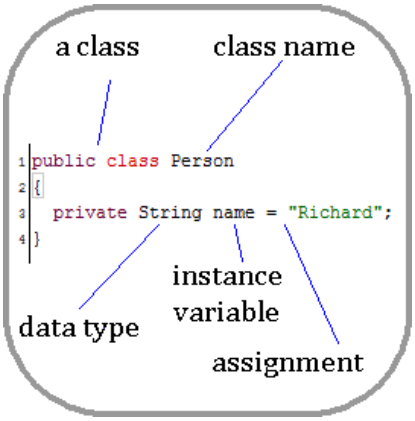
You will notice that the instance variable name has the word `String` before it. This denotes its *data type* – the kind of data that can be stored in the variable. Simple types are text and numbers. Text values are called Strings.

Assignment

Finally, you see that name is given a value. This is called *assignment*. Assignment means giving an instance variable a value.

```
private String name = "Richard";
```

Terminology introduced so far:



An introduction to classes

So, we have a class – it is a definition of some real world object we want to model with our computer program. The world is made up of these objects. Every object has two features that we can try to include (usually in a simplified way) in our computer programs: properties and functions. Name is a property of a person. Our class doesn't have any functions yet but we will consider that shortly.

data type
Data is stored differently to text if it is a number so that arithmetic can be done.

assignment
Putting data in an instance variable.

instance variable
used to store data in a class, we choose the name (there are some rules). Can only hold one data value at a time.

When we create an instance variable with a statement like this we are said to ***"declare"*** *the variable.*

statement
A Java statement is a command or instruction to the computer. It ends with the semi-colon.

When we want to create objects that we can use in our programs we are said to be creating an object instance (or just instance) from a class. The correct way to do this in Java is to use a *constructor*.

Methods

A *method* is a block of statements (sometimes only one) that are put together under a name, so that it can be used (called) when needed. In a typical class there are methods for passing information to a class and for getting information out of the class:

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

```
public String getName(){  return name; }
public void setName(String newName){  name =
newName;  }
```

> ***method***
> *a convenient way to group statements together under a single name.*
>
> ***return type***
> *This indicates that a method returns a value – we will shortly see this in operation.*

We choose the names for the methods and, by convention, they start with a lowercase letter. Any internal words have capitals (sometimes called CamelCase or BumpyText). This makes longer names more readable for humans. .

As for the class, *public* means that other classes can use the method (if it was *private*, they could not). The *String* keyword means that the getName() method returns a String value to the place it was called from (more later). If the method has a return value then a return statement is also needed.

```
public String getName(){  return name; }
```

Some methods do not have return types. For example the method setName is declared as `void`:

```
public void setName(String newName){  name = newName;  }
```

The setName() method also has a *parameter* inside the brackets following the method name – the parameter is used to pass information into the method. The keyword `void` indicates that it does not return any values.

> ***parameter***
> *A method may have one or more parameters used to pass information to the method.*

Our class now has functions as well as a property:

`name`	is a property of a Person
`getName()`	can tell us what the name is
`setName(String newName)`	can assign a value to name

Our class now looks like this:

```
public class Person
{
  private String name;
  public String getName(){  return name; }
  public void setName(String newName){  name = newName;  }
}
```

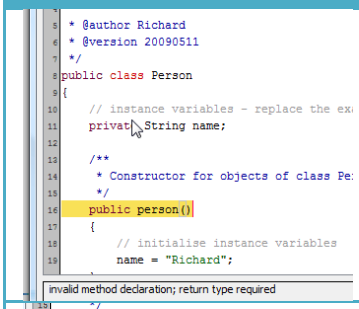
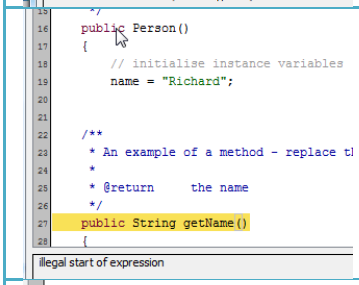
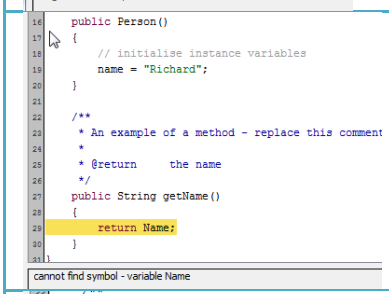
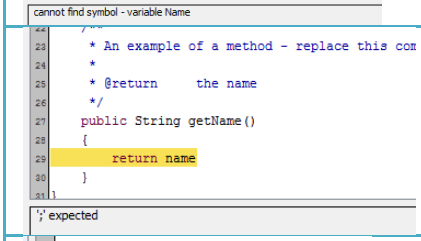
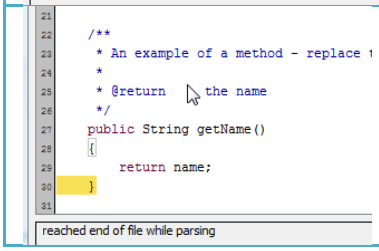
Note: We no longer assign the name directly to the instance variable, we use the setName method. This will allow us to have instances of the Person class with different names as we shall see. First let's try compiling our class:

© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Compiling

Click the compile button, if a message appears near the bottom of the screen *class compiled – no syntax errors* then that is excellent progress. However, you won't be the first programmer this does not happen to. Quite often you will see an ugly yellow highlight and an error message of some kind. Usually these error messages are confusing to beginning programmers.

Some beginner problems

This	Means this
	The constructor name must be the same as the class name (Person) – with a capital letter.
	Check did you leave a closing bracket off the previous method or constructor.
	Message was spelled with a lowercase letter before. Cannot find symbol is usually a spelling mistake. Also check that you haven't confused a 1 with an l or other similar characters.
	The semi-colon is missing from the end of the statement.
	You left out the closing brace of the class.

I'm sure you will find ones not mentioned here as well.

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Main programs

There are a few ways to run computer programs written in Java. We can output text at the console or we can build graphical user interfaces (GUI's). To start with we are just going to output text at the console, when we have a few basic principles established we will do more interesting things with graphical objects (like the boxes and buttons you are used to using).

To create a program in Java we will use another class, one that has only one method, called main:

```
public class SayHello
{
  public static void main(String[] args)
  {
      Person myPerson = new Person();
      myPerson.setName("Harold");
      String name = myPerson.getName();
      System.out.println("Hello " + name);
  }
}
```

We have seen something like the first two lines before. We define a new class called `SayHello`. We have no data members.

Then we have a special method called main. Main always has this format for historical reasons which we won't go into here (you can always Google it).

The next line constructs a new instance of class Person:

```
Person myPerson = new Person();
```

I
The next line calls the method `setName(String newName)` inside class `Person`:

```
myPerson.setName("Harold");
```

Having set the name, we get it back into a variable:

```
String name = myPerson.getName();
```

The following line prints a greeting to the screen:

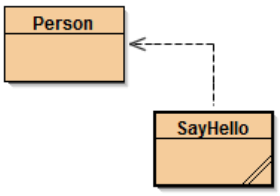
```
System.out.println("Hello " + name);
```

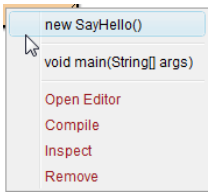
The + sign in the middle of the brackets is an operator that joins two Strings together (concatenates).

If you compile SayHello this happens:

calling a method
The method getName in the class Person is being "called" here. Whatever it returns is assigned to the String name in the SayHello class.

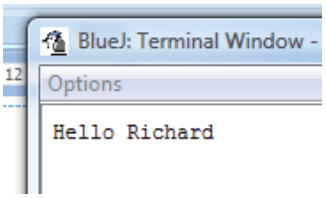
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.





The two classes link up – SayHello is using class Person. If you now right-click SayHello you can run the main method directly. Click the `void main(String[] args)` part, then OK and the console ought to output the message:

```
Hello Harrold
```



(Or whatever you chose to pass as a name).

This is your first Java program.

Why make things so complicated?

Clearly, what we see here is overkill in terms of getting a message printed on the screen. In Java as in most other programming languages, this can be done much more simply.

The use of classes as definitions or blueprints is central to Object Oriented Programming – it combines attributes (or properties) of abstract objects together with the actions (or methods) that define the object's behaviour.

As programs get more complex, the OOP approach encapsulates properties and actions together in a single unit that can be built upon (extended) as necessary.

Example of a useful class

We have actually already been using a more useful class without making it obvious. String is a class that handles character strings. A character string is a sequence of Unicode characters and there are various methods that can operate on these strings. Some examples are:

Examples of String data:

```
"Hello World!"

"the cat sat, nervously, on the mat"

"221b Baker Street"
```

Notice that the start and end of the `String` is delimited by quotation marks, this allows us to pad the Strings with spaces if we want to:

```
"   the cat sat, nervously, on the mat   "
```

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Examples of String methods:

```
String s1 = "Hello World!";
String s2 = " ell ";

System.out.println(s1.equals(s2));
System.out.println(s2.trim());
System.out.println(s1.indexOf(s2));
System.out.println(s1.indexOf(s2.trim()));
```

Output:

```
false
ell
-1
1
```

> ***"dot" notation***
> *To call a method of an instance of a class, place a dot between the instance name and the method name: instance.method().*
>
> *If the method returns a value, you will need a variable to hold it, as seen on the next page.*

A bit more on variables

We mentioned already that variables are memory locations inside the computer.

If we add these statements to our program:

```
String name = "Richard";
name = "Jones";
```

It is important to remember that the first value is completely lost; the computer memory location now looks like this:

Internal reference	Computer Memory	Named instance variable
01110101		
01110110	"Jones"	name
01110111		
01111000		
01111001		
01111010		
01111011		

Every instance variable needs to be given a type. The type describes the kind of data the memory location holds (might be text, numbers, special values). Our instance variable holds `String` (or text) data:

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

A bit more on instances

Once we have a class defined, more instances can easily be created in a program and their methods used with the "dot" form we saw in SayHello:

```
public class MorePeople
{
  public static void main(String[] args)
  {
      // Create two Person instances:
      Person myPerson = new Person();
      Person anotherPerson = new Person();
      // Set their names:
      myPerson.setName("Richard");
      anotherPerson.setName("Huang");
      // Get their names back
      String name1 = myPerson.getName();
      String name2 = anotherPerson.getName();
      // output a message
      System.out.println(name1 + " says hello to " + name2);
  }
}
```

Here we use the class "blueprint" to create two new object instances (`myPerson` and `anotherPerson`)/

The lines in blue, like:

```
      // Create two Person instances:
```

are comments (these are for humans to be able to understand what is being done, the compiler ignores all comments).

Notice that each instance has its own copy of the instance variable name:

```
      myPerson.setName("Richard");
      anotherPerson.setName("Huang");
```

Compile and run the program, if (when) you get syntax errors, refer back to the boxes on page 8 for help.

Assignment 1

Write a new class that prints out the message:

```
Vidyut Hiremath and Moubin Aboualzahab are friends
```

or something similar.

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

The constructor

The constructor is very important in Java, so important that every class has one, whether we put it there or not. For our Person class, Java quietly added a constructor method like this to the class:

```
public Person(){ }
```

Like methods, the *constructor* has braces to mark its start and end. The constructor **always** has the same name as the class and never has any return value (which distinguishes from a method which must always have one, or `void`).

This constructor does nothing obvious; however, it does work "behind the scenes" such as allocating space inside the computer's RAM (Random Access Memory) to store any data belonging to an instance.

In the case of our Person class, the String `name` needs to be stored somewhere the compiler can "tag" it for later use.

One useful use for a constructor is to initialize an instance variable. Here is our class, all nicely commented and laid out easily for humans to read:

```
/**
 * A class to store a Person's name
 */
public class Person
{
  private String name; // The name of the person
  /**
   * Constructor for objects of class name
   */
  public Person(String newName)
  {
      setName(newName);
  }
  /**
   * Returns the value of the name
   */
  public String getName()
  {
      return name;
  }
  /**
   * Sets the value of the name
   */
  public void setName(String newName)
  {
      name = newName;
  }
}
```

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

One thing this change does do is break our existing program, now that we have explicitly added a constructor, we can no longer use the simple constructor:

```
// Create two Person instances:
Person myPerson = new Person();
Person anotherPerson = new Person();
// Set their names:
myPerson.setName("Richard");
anotherPerson.setName("Huang");
```

However, we can now combine these lines into the simpler:

```
// Create two Person instances:
Person myPerson = new Person("Richard");
Person anotherPerson = new Person("Huang");
```

And/or we can add a simple constructor, which is often used to initialize instance variables to some "neutral" value:

```
public Person(){ name = ""; }
```

The complete MorePeople class now looks like this:

```
public class MorePeople
{
  public static void main(String[] args)
  {
      // Create two Person instances:
      Person myPerson = new Person("Richard");
      Person anotherPerson = new Person("Huang");
      String name1 = myPerson.getName();
      String name2 = anotherPerson.getName();
      // output a message
      System.out.println(name1 + " says hello to " + name2);
  }
}
```

A class is loosely called an object and an instance is loosely called an object too. Naturally, this is confusing to beginners.

Class: The class is the template, the blueprint, the definition of the object.

Instance: The instance is created when the constructor is called – it occupies space in the computer's memory.

To use a building analogy, the class is like the blueprint for a house; you can't live in it yet, or sell it or do anything with it. Only when the constructor (the builder) is called and the instance created can you do

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

something with the house. To stretch the analogy a little further, the same blueprint can be used to create many houses.

Anatomy of a class

`public class Person` `{`	Class Name
`private String name;`	Instance variable(s)

`public Person(String newName)`
`{`
`setName(newName);`
`}`
	Constructor

`public String getName()`
`{`
`return name;`
`}`
	Method with return value

`public void setName(String newName)`
`{`
`name = newName;`
`}`
	Method with parameter

`}`	

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Assignment 2

You can have more than one instance variable in a class:

```
public class City
{
    // instance variables – first and last name
    private String name;
    private String country;
```

To complete this class would require:

A constructor named City (logically, this would take two parameters).

Four methods that set and get both instance variables, for example:

```
    public String getName()
    {
        return name;
    }
```

Create the class and a main program that outputs:

```
Paris is in France
New York is in the USA
```

for example.

Help with getting started on this assignment is included in the course.

Javadoc Comments:

When you create a new Class in BlueJ, you see comments in a certain style. As we said before, the comments are for humans, the compiler simply ignores them (they don't get turned into bytecodes at all).

```
/**
 * A class that keeps details of a person
 *
 * @author Richard
 * @version 20090511
 */
```

It is an excellent habit to write comments for yourself and for people that want to help you with your programming.

The @ symbol is useful as it can help document your classes automatically.

SOME RIGHTS RESERVED
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Review of the concepts covered so far

We have covered a lot of ground in a very short time so you are probably quite confused. This is perfectly normal.

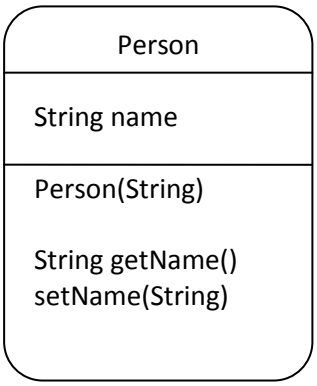
Computer programming attempts to "model" the real world in some way. To do this it needs to represent *objects*. Objects have *properties* and *functions*. Think of people, people in the abstract have a set of qualities or properties, the most commonly used one might be their name. The *instance variables* of a class store those properties.

A *class* can define people in the abstract. To deal with a real person we need to create an *instance* (object) based on that class. People and other objects also typically have functionality – occasionally a person wants to change their name. The *methods* of a class define that functionality. Here is a compact version of a Person class:

```
public class Person                                    // class
{
  private String name;                                 // instance variable
  public Person(String newName)                        // Constructor
  {
    setName(newName);
  }
  public String getName(){ return name; }             // Method
  public void setName(String newName)                  // Method
  {
     name = newName;
  }
}
```

There really isn't that much to it after all.

Useful classes will have more instance variables and more methods. Pictorially we might represent the class like this:



class name	definition
instance variable	property
constructor	create instance
methods	functions

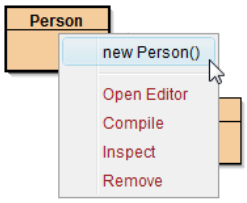
© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.

Using BlueJ to interact with Object Instances

One of the nice features of BlueJ is that we can actually create and alter objects on the Workbench. If you right-click on the Person class you will get a menu, one of the items in that menu is new Person(). If you click this, you will call the constructor and be asked what name you want to give to the instance.

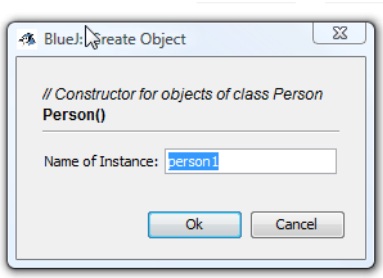
Once the object is on the Workbench you can "inspect" its properties, for example you can call the method getName() by clicking it.

The steps are shown below:

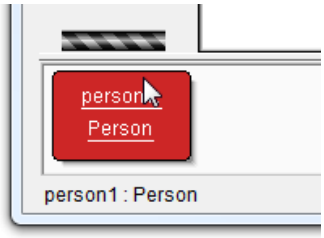


Right-click over the class name

Click new Person()

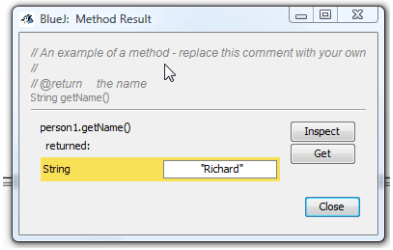


You can accept the suggested name or change it



An instance is placed on the Workbench

The instance is called person1 (unless you renamed it)



If you right-click the person1 instance, you can run the method getName() to see the value of the name instance variable.

© 2010 Richard Jones, 347 Ariki Street, Karapiro Village, Cambridge 3495, New Zealand. richardnz@moodlenz.net
This work is licensed under a Creative Commons Attribution-Non-commercial-Share Alike 3.0 New Zealand License.